An Introduction to R

Contents

1 Introduction

These notes provide a simple introduction to R. Further instruction will be provided throughout various modules, but a good way to learn R is to experiment for yourself.

1.1 What is R?

R is a powerful interactive computer package that is orientated towards statistical applications. It will run on the most commonly used platforms (or operating systems) Windows, Linux and Mac. The notes here are orientated towards use on a Windows platform. It consists of a *base system* that can be downloaded without charge together with many contributed packages for specialist analyses. It offers:-

- an extensive and coherent set tools for statistics and data analysis

- a language for expressing statistical models and tools for using linear and non-linear statistical models

- comprehensive facilities for performing matrix calculations and manipulations, enabling concise efficient analyses of many applications in multivariate data analysis and linear models

- graphical facilities for interactive data analysis and display

- an object-orientated programming language that can easily be extended

- an expanding set of publicly available libraries or packages of routines for special analyses

- libraries or packages available from the official Contributed Packages webpages are thoroughly tested by the R Core Development Team

- packages that have manuals, help systems and usually include illustrative datasets

1.2 R on the campus network

On University Managed Desktop machines R is available under the Statistics applications menu. You will need to first double click the "Load application menus" icon to obtain the Statistics applications menu.

1.3 Installing R on your own computer

Full instructions for installing R are given on the R Project home page at

`http://www.r-project.org/`

The first step is to choose a site geographically close to you from which to download the

package. Next choose the appropriate operating system, select the option `base` and download the system. Accepting the option to run the download file will install the package after downloading it. Accepting the default options for locations etc is simplest but these can be customized. By default an icon (with version number) is placed on the desktop.

1.4 The R Graphical user interface (RGui)

When you open an R *session* (i.e. start the R program), the R Graphical user interface opens and you are presented with a screen like this:

```
R version 2.9.0 (2009-04-17)
Copyright (C) 2009 The R Foundation for Statistical Computing
ISBN 3-900051-07-0

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

>
```

Along the top of the window is a limited set of menus, which can be used for various tasks including opening, loading and saving script windows (section 5) loading and saving your workspace (section 6.1), and installing packages.

The symbol `>` is the *command line prompt* symbol. Typing a command or instruction at the prompt will cause it to be executed immediately.

1.5 Online help

R has various online manuals that can be accessed from the **help** menu on the menu bar. You can get help on specific commands by typing ? followed by the command name. For example, for help on plotting graphs, try

```
?plot
```

1.6 Quitting R

You can quit R by typing `q()`. You will be asked if you want to save the workspace image. See section 6.1 for more details.

2 Simple R functions

All commands in R are regarded as *functions*, they operate on *arguments*, e.g. `plot(x, y)` plots the vector `y` against the vector `x` - that is it produces a scatter plot of `y` vs. `x`. It is useful to think of the commands that you can give R in three categories: evaluation, assignment, and display.

2.1 Evaluation

To evaluate a function, just type its name followed by the list of arguments, which may be empty, given in brackets. For example, `sqrt` is the name of the function that calculates square roots: the command

```
sqrt(3)
```

returns

```
[1] 1.732051
```

Most functions produce numerical (or other) output, such as the `sqrt` example above. For example, the function `rchisq` generates a random sample of specified size from a chi-squared distribution—try

```
rchisq(20,5)
```

to generate a sample of size 20 from a chi-squared distribution with 5 degrees of freedom.

An expression to be evaluated need not be in the form of a function call; the usual arithmetic operators `+`, `-`, `*`, `/`, and `^` are all available, as are a number of more specialized operators. Try calculating, say, `((3+1)^(1/2))/2`.

2.2 Assignment

In Section 2.1, the results of the evaluations are displayed immediately. Instead, the results of an evaluation can be assigned, invisibly, to a variable, as in

```
x <- rbinom(10,5,0.2)
```

which creates a vector of 10 binomial random variables (each with 5 trials, and probability of success 0.2), and calls it x. In fact there are three equivalent symbols for assignment, =, _ and <-, and the spaces round each are optional. If x had already been defined by the user, it would have been over-written. To assign numerical values directly, use the format

```
y <- 3
```

to set y to the value 3.

Functions can be performed on variables. For example,

```
max(x)
```

returns the largest element of the vector x.

Numbers can also be read in from files, or calculated, constructed or randomly generated by many different functions. For example, `rep(x,n)` gives n repetitions of x, where x can be almost anything, and `seq` constructs evenly spaced sequences: try

```
seq(1,99,3)
```

Remember, assignments do not automatically display anything.

Variable (and command) names are case-sensitive, and you should avoid using names that also serve as R functions, for example, t, c, q, F and T.

2.3 Display

To see the value of an object that you have created, or that already exists in R, just type its name; try just typing x (and pressing **Return**). This applies equally to functions; to R, functions are just objects of a particular sort. This is why it is necessary to include the empty argument list () with the command q in order to quit; just typing the name of the function would result in the function itself being displayed. Try typing q without the brackets.

To see the names of all the variables you have defined in your session, type

```
ls()
```

3 Vectors and matrices

3.1 Vectors

A vector of numbers can be made using the c command, for example

```
x<-c(2,4,6,8,10)
```

Various commands can be performed on vectors such as `sum(x)` and `mean(x)`. Individual elements of vectors can be accessed by specifying the required index within square brackets:

```
x[2]
```

gives the second element of x, and

```
x[c(1,2,3)]
```

gives the first three elements of x.

Operations can be performed on each element of vector using a single command. For example, `3*x` or `x^2`. Element-wise operations between vectors can be performed similarly. For example, given `y<-c(1,3,5,7,9)`, and x as defined above

`y*x` returns the result

```
[1]  2 12 30 56 90
```

3.2   Matrices

Matrices are defined using the `matrix` command. For example,

```
m<-matrix(c(2,4,6,8,10,12),nrow=3,ncol=2)
```

will create the matrix
$$\begin{pmatrix} 2 & 8 \\ 4 & 10 \\ 6 & 12 \end{pmatrix}.$$

By default, R will fill the matrix with the specified data one row at a time. If you add the argument `byrow=T`

```
m<-matrix(c(2,4,6,8,10,12),nrow=3,ncol=2,byrow=T)
```

you obtain
$$\begin{pmatrix} 2 & 4 \\ 6 & 8 \\ 10 & 12 \end{pmatrix}.$$

The command `t(x)` returns the transpose of the matrix x, and if x is a square, invertible matrix, its inverse can be calculated using `solve(x)`.

Elements of matrices are selected using square brackets:

```
x[3,2]
```

specifies the element in the 3rd row, 2nd column. Entire rows and columns can also selected. `x[2,]` gives the 2nd row and `x[,1]` gives the first column.

Both element-wise and standard matrix multiplication can be done. The command

```
m*m
```

does element-wise multiplication to produce

$$\begin{pmatrix} 4 & 16 \\ 36 & 64 \\ 100 & 144 \end{pmatrix}.$$

Standard matrix multiplication for two conformable matrices `x` and `y` is done with the command

```
x%*%y
```

4   Plotting Graphs

4.1   Basic plots

Suppose you have two vectors:

```
x<-c(1,2,3,4)
y<-c(1.1,0.5,4,6)
```

To produce a scatter plot of `y` against `x` type

```
plot(x,y)
```

Adding the argument `type="l"` produces a line plot

```
plot(x,y,type="l")
```

4.2   Specifying axes labels

You can specify your own axes labels and title with additional arguments

```
plot(x,y,xlab="x-axis label", ylab="y-axis label", main="Plot title" )
```

Font sizes can be changed by first typing

```
par(ps=15)
```

before plotting, modifying the number after `ps` as desired.

4.3 Multiple plots on the same axes

To add more points or lines to an existing plot, for example, corresponding to two more vectors x2 and y2, use the `points` or `lines` commands

```
points(x2,y2)
lines(x2,y2)
```

A useful command for plotting functions is `curve`. For example, to plot the curve $y = e^{-x^2} + \cos x$ for x between -2π and π, type

```
curve(exp(-x^2)+cos(x),from=-2*pi,to=2*pi)
```

To add the curve to an existing plot, include the argument `add=TRUE` in the `curve` command.

4.4 Setting axes ranges

The ranges for both axes are chosen automatically, and are not altered even if you add points or lines that don't fit. If you wish to specify the range for an axis, e.g., you want the y-axis to range from -5 to 10, you should instead type

```
plot(x,y,ylim=c(-5,10))
```

The x-axis range can be specified by adding an `xlim=` argument, e.g.

```
plot(a,b,ylim=c(-5,10),xlim=c(-3,8))
```

(You cannot change the axis ranges when using the `lines` command).

4.5 Copying R graphs to Word

To copy a graph into Word, first make the graph the active window. Then select **File** > **Copy to the clipboard** > **as a Bitmap** . You can now paste the graph into a Word document using the **Edit** menu in Word.

5 Script windows

If you intend to use a series of commands during a session, you should use a script window, as you will find it easier to correct any mistakes or make changes to your commands, and you can save your work afterwards.

To open a script window, select **File** > **New script** from the menu bar. To run any command or selection of commands, highlight the desired commands using the mouse and press

Ctrl+R.

5.1   Saving and loading scripts

To save a script, select the script window and go to **File** > **Save as...** on the menu bar. When entering a filename, it's best to add the `.R` extension at the end, as it will make the script easier if you wish to load it. Scripts can be loaded by selecting **File** > **Open script...** on the menu bar.

6   Saving and loading data

It's usually easiest to work in the same directory (folder) as your data. To change the working directory by selecting **File** > **Change dir...** and then browse for the desired folder. You can see which directory you are currently working with the command

```
getwd()
```

6.1   Saving and loading the workspace image

The workspace image consists of all the variables (and user-defined functions) that you have defined during your session. You can save the image at any time by selecting **File** > **Save workspace...** on the menu bar, and then choosing a file name and location. You can load a previously save image at any time by selecting **File** > **Load workspace...** on the menu bar

You will also be asked if you want to save the workspace image when you quit R. If you choose to save it, the image will be automatically loaded the next time you start R.

6.2   Loading an existing R dataset

Load an existing R dataset with the command

```
load(" ")
```

inserting the filename inside the quotes. You will need to specify the full path, unless you have changed the working directory as described above.

6.3   Saving specific variables

If you only want to save specific variables rather than the whole workspace image, you can do so with the `save` command. Again, it's usually easiest to set the working directory to be the desired destination folder for saving.

This example creates one vector and one matrix, and saves them under the single filename `mydata.Rdata`

```
x<-c(2,4,6,8,10)
M<-matrix(c(1,0,0,1),nrow=2,ncol=2)
save(x,M,file="mydata.Rdata")
```

6.4   Importing data into R

6.4.1   Plain text (tab separated) format

You may wish to import data stored in alternative file formats. For example, suppose you have a plain text file `measurements.txt`, with each row giving the age, height, and weight of an individual, and the first row giving the column headings:

```
age     height  weight
22      172     68
25      150     60
32      180     75
```

You can import this into R and store it as a *data frame* called `measurements` with the `read.table` command

```
measurements<-read.table("measurements.txt",header=T)
```

(assuming the file `measurements.txt` is in the current working directory). You can extract single columns from the data frame as follows:

```
measurements$age
measurements$height
measurements$weight
```

The option `header=T` tells R to use the first line of the text file as the column names. If this is changed to `header=F`, R will assign its own column names, and all values in the text file will be used as data values, whether this is appropriate or not. For example

```
read.table("measurements.txt",header=F)
```

produces

```
    V1      V2      V3
1 age  height  weight
2  22     172      68
3  25     150      60
4  32     180      75
```

so the column names are now `V1`, `V2`, `V3`, with `age` the first element of column `V1` and so on. Hence you should use `header=T` if the first row of your text file gives the column names, and `header=F` if your text file contains data values only.

6.4.2   Excel datasets: comma separated values format

If you wish to import data from Excel, you should first save the Excel file as a CSV file. You can then use the command `read.csv`, for example,

```
measurements<-read.csv("measurements.csv",header=T)
```

Again, you should specify `header=T` or `header=F` depending on whether the first row in your dataset specifies the column names or not.